Введение в реактивное 🚀 программирование

Сэм Булатов



Давайте знакомиться

- 🥸 Сэм Булатов
- 🫠 4+ лет фронтенд-разработки
- 🗺️ Фронтендер в Waliot
- 👥 Организатор в krd.dev

О чем будем говорить

- Что такое реактивное программирование?
- Какую проблему пытается решить этот подход?
- Как это все связано с фронтендом и фреймворками?

Краткая история

90 г.

- Постоянная перепроверка состояния по cooldown
- Вследствие этого может быть задержка отклика от интерфейса

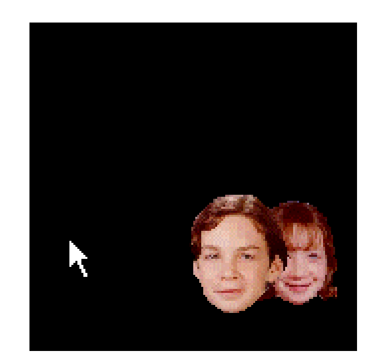
92-95 гг.

- В 1992 году Microsoft представляет новый подход к разработке интерфейсов — Event Driven
- Позже, в 1995, этот шаблон проектирования назовут Observer

97 г.

FRAN (Functional Reactive Animation)

```
kids u =
  delayAnims 0.5
  (map (move (mouseMotion u))
  [jake, becky, charlotte, pat])
```



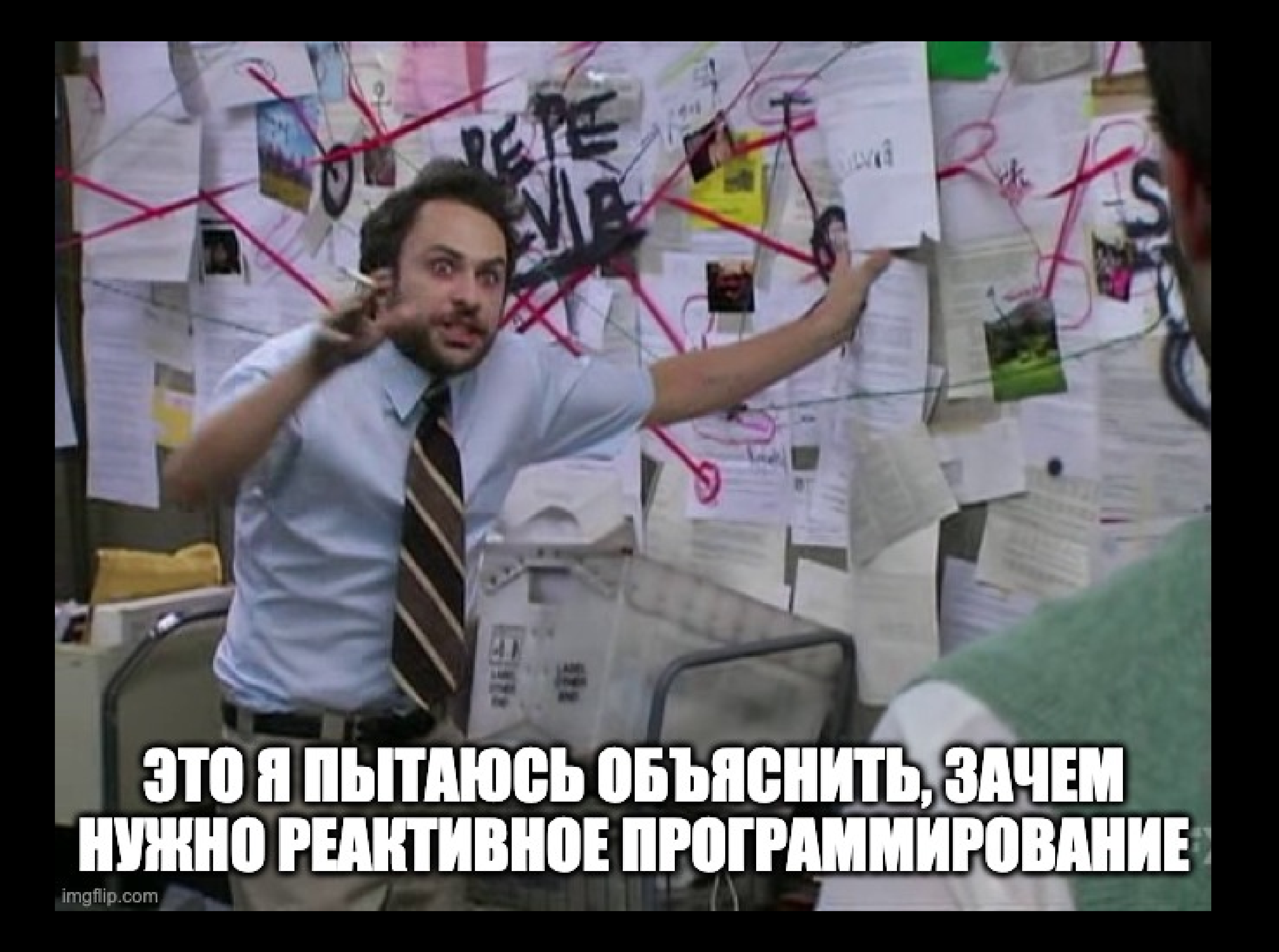
00-е



У сети много проблем, которые надо решать

- Сеть это задержка, данные получаются немоментально
- Сеть может упасть, она нестабильна
- У сети есть ограничения, например, пропускная способность
- У устройства пользователя тоже есть ограничения, например ОЗУ
- и еще много всяких разных неприятностей...

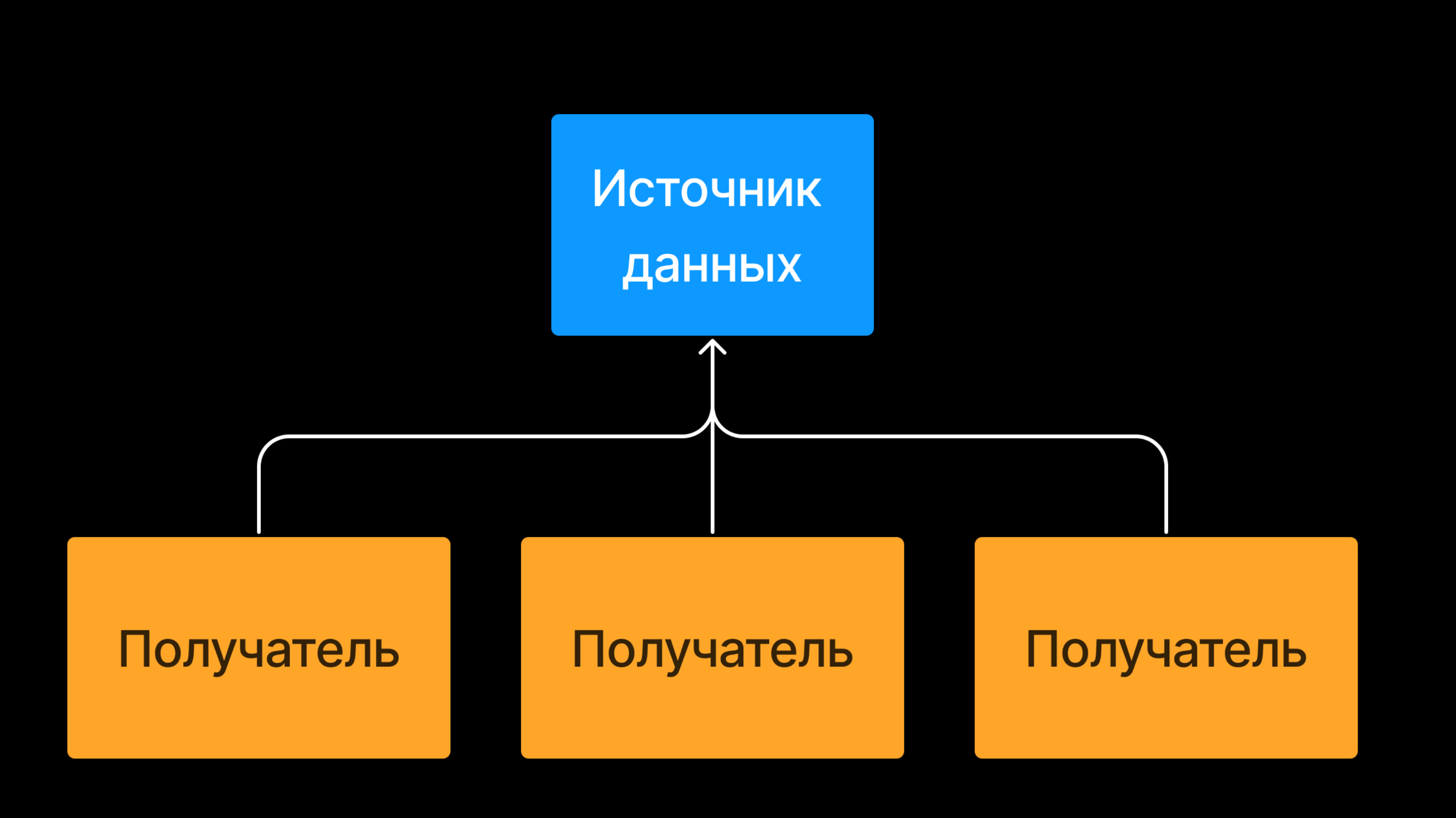
Reactive Extension

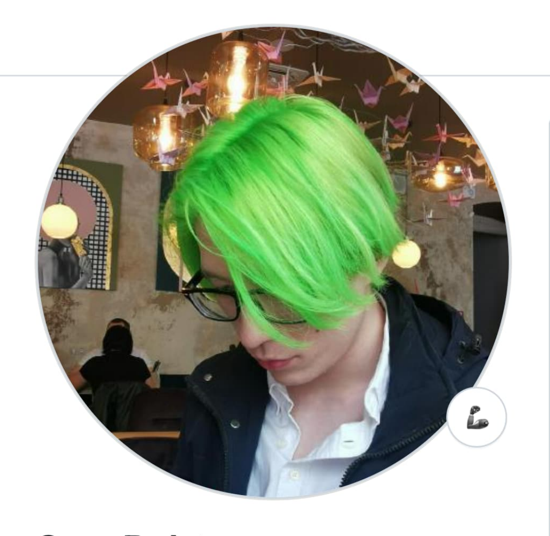

ЭТО Я ПЫТАЮСЬ ОБЪЯСНИТЬ, ЗАЧЕМ
НУЖНО РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ
imgflip.com

Как только ты понимаешь, что такое реактивное программирование, ты теряешь способность объяснить это другим.

Кто-то очень умный

RxJS?


Источник данных
Получатель
Получатель
Получатель



Sam Bulatov

mephistorine

Frontend developer at **@waliot** and **@nible-io**. Member of the organizing committee of the **@krddevdays**. Core editor at **@learnrxjs**.

Edit profile

52 followers · 125 following

@waliot, @nible-io

Russia, Krasnodar

mephistorine / README.md

Hi, I'm Sam Bulatov 👋



Logo

I am Frontend developer at @waliot and @nible-io also core maintainer @be-Cycled app

- I'm looking to collaborate on @be-Cycled
- All of my projects are available at https://mephi.dev/projects
- I regularly write articles on https://mephi.dev
- Ask me about **angular, rxjs**
- How to reach me **mephistorine@gmail.com**
- Know about my experiences https://mephi.dev/resume
- Fun fact **Professional rollerblade skater**

Connect with me:

- twitter
- instagram

Languages and Tools:

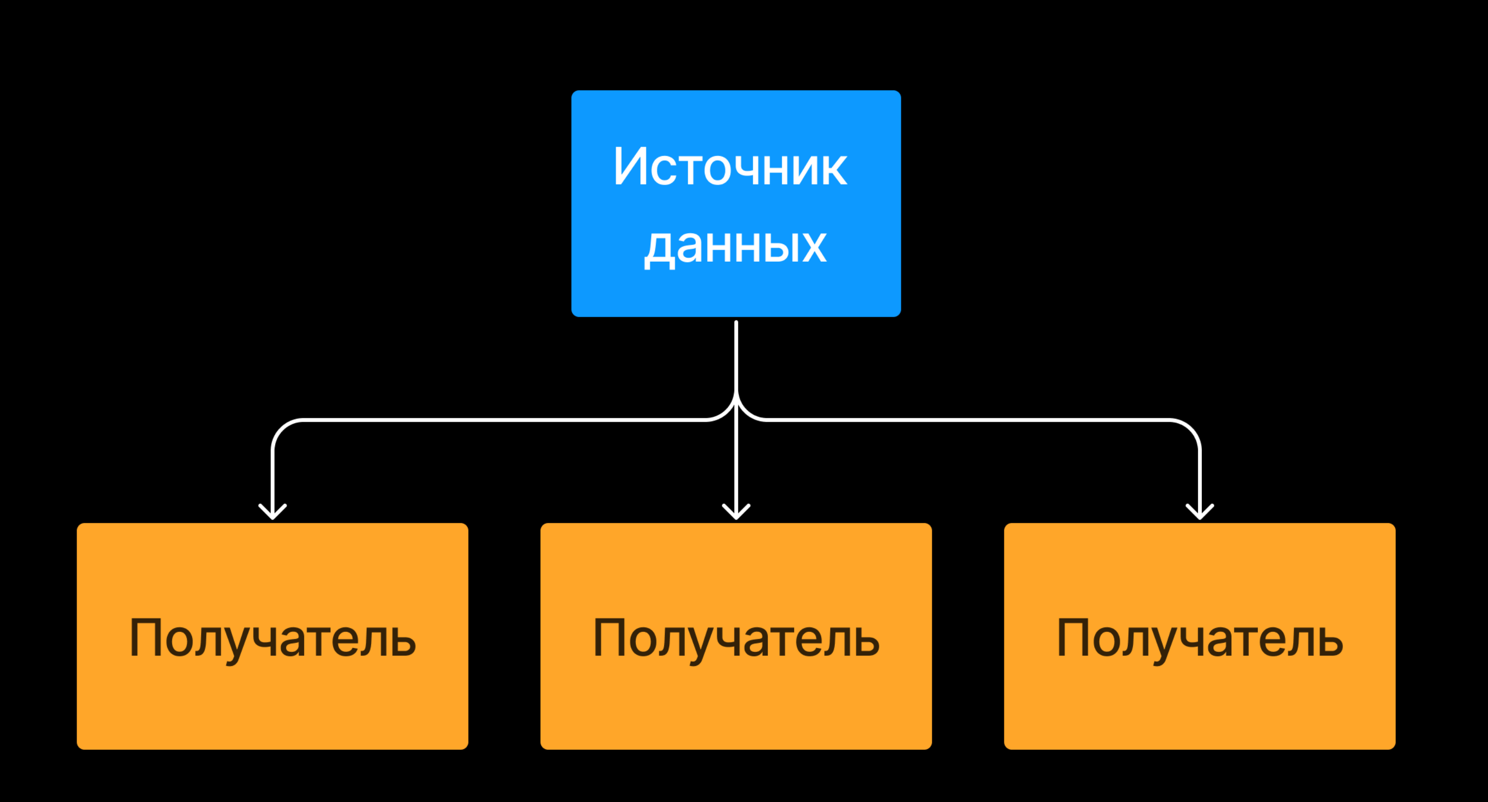
- angular
- typescript

Main projects

- LearnRxJSRu – Unofficial russian translate RxJS documentation and more

Maintained chats

- RxJS — русскоговорящее сообщество


Источник данных
Получатель
Получатель
Получатель

Абсолютно

Реактивность !== Асинхронность

click

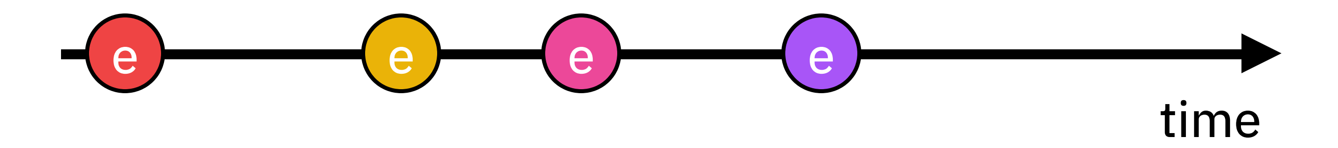
mousemove

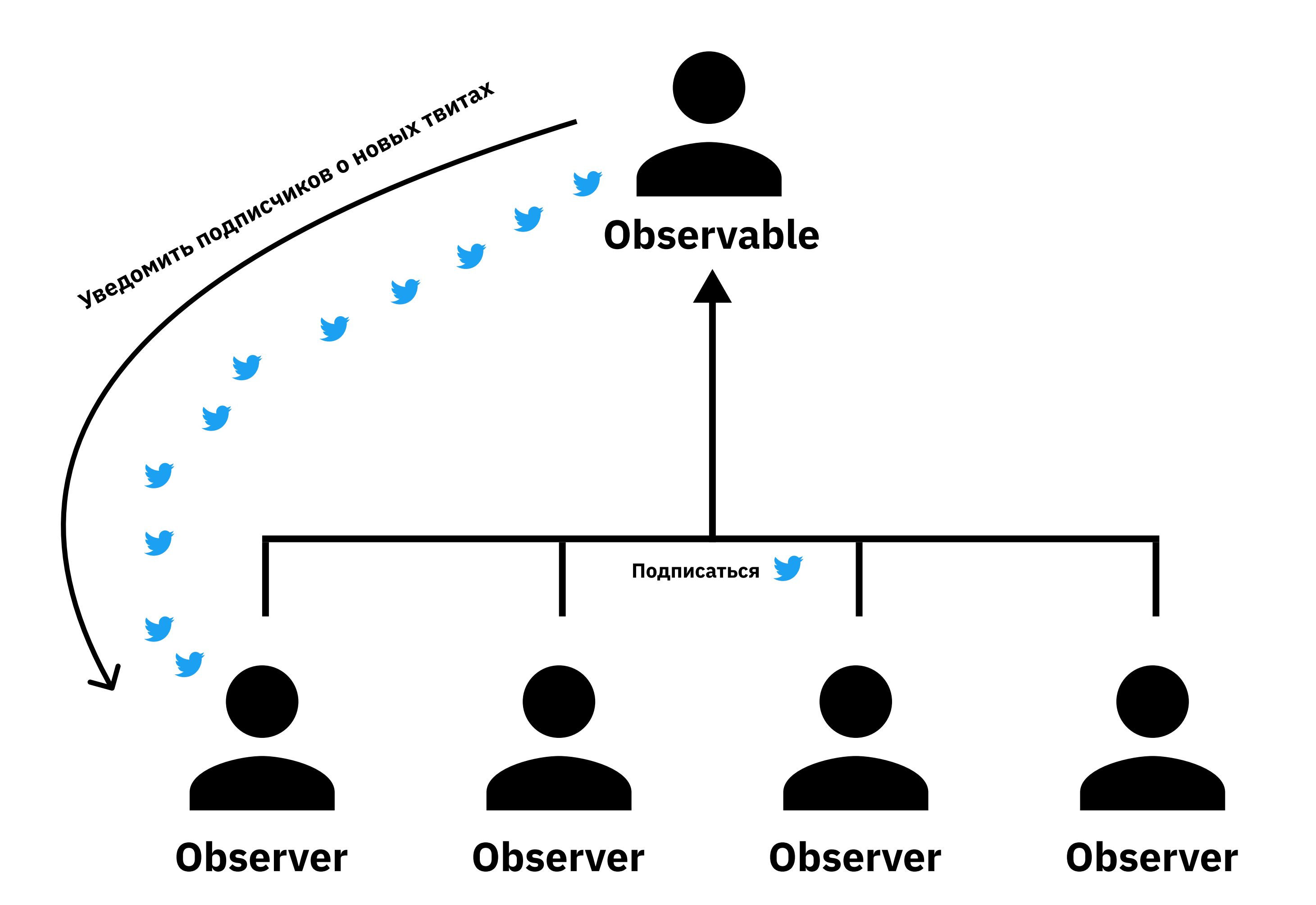
load

WebSocket

DataStructure

Array


e
e
e
e
time



Уведомить подписчиков о новых твитах
Observable
Подписаться
Observer
Observer
Observer
Observer

Принципиальная разница в модели получения данных

Модель описывает модель взаимодействия меж двумя сущностями, **Производителем** данных и их **Потребителем.** Моделей всего две:

- Pull (Вытягивание)
- Push (Проталкивание)



	Единичное	Множественное
Pull	Function	Iterator
Push	Promise	Observable

Пример из жизни

Шаги

Калории

```
isGoalReached = stepCount > X &&
caloriesCount < N
```

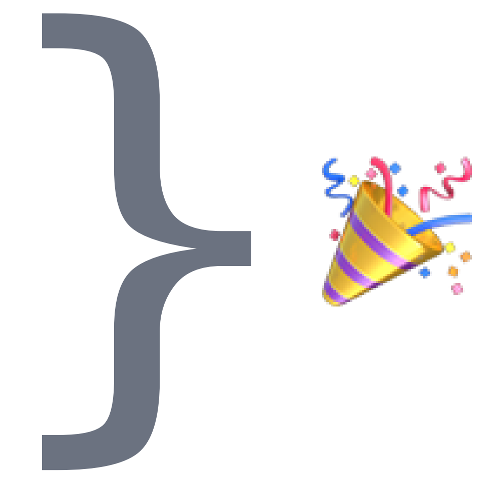
`isGoalReached`

Observable **Observer**



4960 4970 4980 4990 5000 ...

Поток данных: Шаги





90 90 110 120 130 ...

Поток данных: Калории



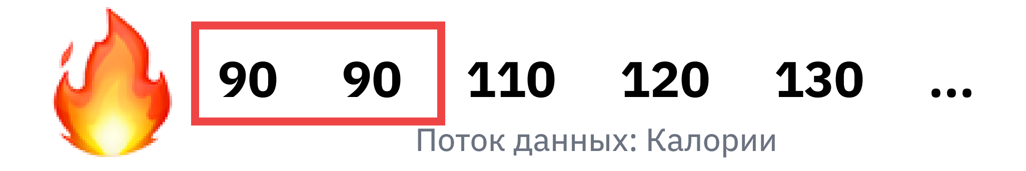
(4960,90) (4970,90) (4980,110) (4990,120) (5000,130) time

(4960,90) (4970,90) (4980,110) (4990,120) (5000,130) time

```
filter(логика проверки цели)
```

(5000,130) time

Congratulations🎉


90
90
110
120
130
...
Поток данных: Калории

90 90 110 120 130

time

`distinct()`

90 110 120 130

time

Реализации реактивности

Объектные

- CellX
- MobX
- $mol_wire

Функциональные

- Preact Signals
- RxJS
- Effector
- nanostores

Preact Signals и RxJS

RxJS



```
fromEvent(document, "click")
  .subscribe((event) => console.debug(event.type))
```

```
fromEvent(document, "click")
  .pipe(
    scan((count) => count + 1, 0)
  )
  .subscribe((count) => console.debug(`Count: ${ count }`))
```

Операторы

- combine
- filter
- distinct

Это не магия

```
const multipleX2 = (value) => value * 2
const formatNumber = (value: number) =>
                value.toLocaleString("RU-ru")
const produceNum = compose(multipleX2, formatNumber)
console.log(produceNum(22_433.33))
// > 22 433,3
```

Это не магия

```
.pipe(
  map(),
  filter(),
  distinctUntilChanged()
)
```

Реализуем на RxJS

```
function isGoalReached(
  currentStepCount: number,
  currentCalories: number
): boolean {
  return currentStepCount >= 5000 && currentCalories >= 130
}
```

```
const steps = from([ 4960, 4970, 4980, 4990, 5000 ]).pipe(
  concatMap((value) => of(value).pipe(delay(1000)))
)
```

```
const calories = from([ 90, 90, 110, 120, 130 ]).pipe(
  concatMap((value) => of(value).pipe(delay(1000)))
)
```

```
combineLatest([ steps, calories ])
    .subscribe(() => sendCongratulationsNotify())
```

```
combineLatest([ steps, calories ])
  .pipe(
    filter(([ stepCount, calories ]) => isGoalReached(stepCoun
  )
  .subscribe(() => sendCongratulationsNotify())
```

```
const isGoalReachedSub = combineLatest([ steps, calories ])
  .pipe(
    filter(([ stepCount, calories ]) => isGoalReached(stepCou
  )
  .subscribe(() => sendCongratulationsNotify())

isGoalReachedSub.unsubscribe()
```

Abort Controller

```
const controller = new AbortController()
fetch("/hello", { signal: controller.signal })
  .then(() => {})
onDestroy(() => {
  controller.abort()
})
```

```
const controller = new AbortController()
element.addEventListener(
  "click",
  () => {},
  { signal: controller.signal }

)
```

```
const controller = new AbortController()
element.addEventListener(
  "click",
  () => {},
  { signal: controller.signal }

)
```

В RxJS это есть давно!

```
const subject = new Subject()
const observable = from(...).pipe(
  takeUntil(subject)
)
```

```
const sub1 = observable.subscribe()
const sub2 = observable.subscribe()
const sub3 = observable.subscribe()
onDestroy(() => {
  subject.next()

  subject.complete()
})
```

Что еще можно делать?

Что еще можно делать?

- debounceTime
- Promise interop
- retry
- retryWhen (offline)
- shareReplay
- catchError
- webSocket
- ... и еще дофига всего

Preact Signals

```
const name = signal("Jane")
const surname = signal("Doe")
const fullName = computed(() => ...)
effect(() => console.log(fullName.value))
// > Jane Doe

name.value = "Sam"
// > Sam Doe
```

```
const steps = signal(0)
const calories = signal(0)
```

```
const stepValues = [ 4960, 4970, 4980, 4990, 5000 ]

for (let i = 0; i < stepValues.length; i++) {
  setTimeout(() => {
    stepValues.value = stepValues[ i ]

  }, i * 1000)
}
```

```
effect(() => {
  if (isGoalReached(steps.value, calories.value)) {
    sendCongratulationsNotify()
  }
})
```

```
const unsubscribe = effect(() => {
  if (isGoalReached(steps.value, calories.value)) {
    sendCongratulationsNotify()
  }
})

unsubscribe()
```

Итоги

Минусы

- Тяжело изучать
- Сложно дебажить
- Не всегда удобно тестировать
- Может быть запутанный код, особенно с операторами RxJS

Спасибо, с вами был Сэм!

mephi.dev/gsrp



